Mixture-of-Modules: Reinventing Transformers as Dynamic Assemblies of Modules

Zhuocheng Gong[1]* **Ang Lv**[2]* **Jian Guan**[3,4]† **Junxi Yan**[3] **Wei Wu**[4]
Huishuai Zhang[1] **Minlie Huang**[3] **Dongyan Zhao**[1]† **Rui Yan**[2]†
[1]Wangxuan Institute of Computer Technology, Peking University
[2]Gaoling School of Artificial Intelligence, Renmin University of China
[3]Tsinghua University [4]Ant Group
{gzhch,zhanghuishuai,zhaody}@pku.edu.cn, {anglv,ruiyan}@ruc.edu.cn
{yanjx21}@mails.tsinghua.edu.cn, aihuang@tsinghua.edu.cn
jianguanthu@gmail.com, wuwei19850318@gmail.com

Abstract

Is it always necessary to compute tokens from shallow to deep layers in Transformers? The continued success of vanilla Transformers and their variants suggests an undoubted "yes". In this work, however, we attempt to break the depth-ordered convention by proposing a novel architecture dubbed mixture-of-modules (MoM), which is motivated by an intuition that any layer, regardless of its position, can be used to compute a token as long as it possesses the needed processing capabilities. The construction of MoM starts from a finite set of modules defined by multi-head attention and feed-forward networks, each distinguished by its unique parameterization. Two routers then iteratively select attention modules and feed-forward modules from the set to process a token. The selection dynamically expands the computation graph in the forward pass of the token, culminating in an assembly of modules. We show that MoM provides not only a unified framework for Transformers and their numerous variants but also a flexible and learnable approach for reducing redundancy in Transformer parameterization. We pre-train various MoMs using OpenWebText. Empirical results demonstrate that MoMs, of different parameter counts, consistently outperform vanilla transformers on both GLUE and XSUM benchmarks. More interestingly, with a fixed parameter budget, MoM-large enables an over 38% increase in depth for computation graphs compared to GPT-2-large, resulting in absolute gains of 1.4 on GLUE and 1 on XSUM. On the other hand, MoM-large also enables an over 60% reduction in depth while involving more modules per layer, yielding a 16% reduction in TFLOPs and a 43% decrease in memory usage compared to GPT-2-large, while maintaining comparable performance. [1]

1 Introduction

Transformer-based language models (Vaswani et al., 2017) have demonstrated remarkable abilities across a wide range of challenging natural language tasks (Bubeck et al., 2023). In addition, the success of Transformer in natural language processing (NLP) is also inspiring innovations in other fields such as computer vision (Peebles and Xie, 2023; Agostinelli et al., 2023) and biomedicine (Singhal et al., 2023; Madani et al., 2023). A Transformer architecture typically consists of stacked layers that are identical in structure, whereby layers are organized in the order of depth, using the output of the previous layer as the input for the next. While this design convention has been widely accepted as a matter of course in the Transformer era, we seek to challenge it by reconsidering whether the static and depth-ordered organization can fully unleash the potential of Transformers, given their well-known issues with over-parameterization (Zeng et al., 2023) and efficiency (Raposo et al., 2024).

Before us, some rudimentary studies have touched on the question–they dissect Transformer into modules such as attention heads and feed-forward networks (FFNs) and allow relatively flexible module call order. For example, Mixture-of-Experts (MoE) (Shazeer et al., 2017)) sets up multiple FFNs within the same layer and activates a specific subset during inference. Early-exiting (Zhou et al., 2020; Xin et al., 2020; Schuster et al., 2022) and Mixture-of-Depths (MoD) (Raposo et al., 2024)) bypass certain layers when computing each token. On the one hand, these efforts indeed lead to improvements in terms of either efficacy or efficiency through the introduction of dynamic mechanisms into the vanilla structure of Transformers, and thus corroborate our questioning regarding the established convention; on the other hand, they still follow the depth-ordered paradigm (i.e., tokens are passed from shallow layers to deep

*Equal Contributions.
†Corresponding authors.
[1]Code is available at https://github.com/gzhch/Mixture-of-Modules

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 20924–20938
November 12-16, 2024

layers), leaving significant room for better architectures.

In this work, we completely disrupt the traditional practice in the design of Transformers by breaking down the depth-ordered organization. Numerous studies have indicated that knowledge in Transformers is often dispersed across multiple FFNs in different layers (Geva et al., 2021; McGrath et al., 2023; Lv et al., 2024), and many attention heads serve similar or identical functions, such as copying specific token information towards the end position of the input (Olsson et al., 2022; Wang et al., 2023). Encouraged by this evidence, we pose the question of whether the computation of a token can "move" freely across layers, that is the token can be computed by flowing to modules in deeper layers, sticking to modules of the same layer, or even going back to modules in previous layers. To answer the question, we propose a novel architecture dubbed Mixture-of-Modules (MoM) in which the core idea is to define a neural network as dynamic assemblies of modules derived from vanilla Transformer, as depicted in Figure 1.

The basis of MoM is a finite set of modules. Each module is defined by multi-head attention (MHA), a feed-forward network (FFN) (including Add & Norm), or a specialized module labeled "SKIP". Each MHA or FFN module is identical in structure and different in parameterization. SKIP enables skip operations for arbitrary tokens at arbitrary time steps. Given a token, each time two routers select modules from the set and integrate the modules into the computation graph during the forward pass. Hence, the whole computation graph of the token is formed as an assembly of modules, and the routers learn to optimize the organization of the modules in the assembly. We introduce a two-phase approach for training MoM models. In the first phase, we pre-train a vanilla Transformer on a large-scale corpus. Then, in the second phase, we decompose the pre-trained Transformer into modules as a warm-up of MoM, randomly initialize the routers, and further update both the modules and the routers under the mechanism of dynamic assembly.

MoM has three major advantages over existing Transformer-based architectures: (1) it provides a unified framework for various Transformer variants, incorporating popular methods such as mixture-of-experts, early-exiting, and mixture-of-depths as special cases. The framework sheds light on architecture design for future works; (2) it brings unprecedented flexibility in forward computation. With the dynamic assembly mechanism, the conventional concept of "depth" now has a new interpretation: it refers to the number of routings performed per token. Useful modules such as specific attention heads can be repeatedly utilized as long as they are capable of processing. Hence, MoM offers a dynamic and learnable approach to reducing redundant parameters in Transformers; and (3) it offers efficient structures that achieve performance comparable to vanilla Transformers but require significantly fewer FLOPs and less memory in forward computation.

We pre-train MoM in three sizes–122M (small), 346M (medium), and 774M (large)– using OpenWebText (Gokaslan and Cohen, 2019), and assess their performance with GLUE benchmark (Wang et al., 2018a) and XSUM (Narayan et al., 2018b). Empirical results indicate that (1) MoMs consistently outperform vanilla GPT-2 models on both text understanding and generation tasks; (2) parameters are quite redundant in vanilla Transformers. MoM can hold the comparable performance after removing 50% of the MHA modules and 25% of the FFN modules; and (3) for those concerned with efficiency, MoM-large can reduce TFLOPs by 16% and memory usage by 42% in forward computation, while maintaining comparable performance to GPT-2 (774M), via properly increasing the number of modules and compressing the model depth.

Our contributions are three-fold: (1) proposal of Mixture-of-Modules to disrupt the depth-ordered convention in Transformer construction, and reinvent Transformers as dynamic assemblies of modules; (2) empirical verification of the efficacy of Mixture-of-Modules on GLUE benchmark and XSUM; and (3) a series of new insights into the over-parameterization issue of vanilla Transformers, and their implications for future architecture design.

2 Related works

MoM owns a dynamic mechanism of module selection and combination, and thus is related to conditional computation techniques (Bengio et al., 2013; Davis and Arel, 2014; Cho and Bengio, 2014). Existing work on conditional computation can be categorized into two groups: *dynamic depth* and *dynamic width*. In these fields, terms such as gating and routing are used interchangeably, hereafter referred to as "routers" for clarity in presentation.

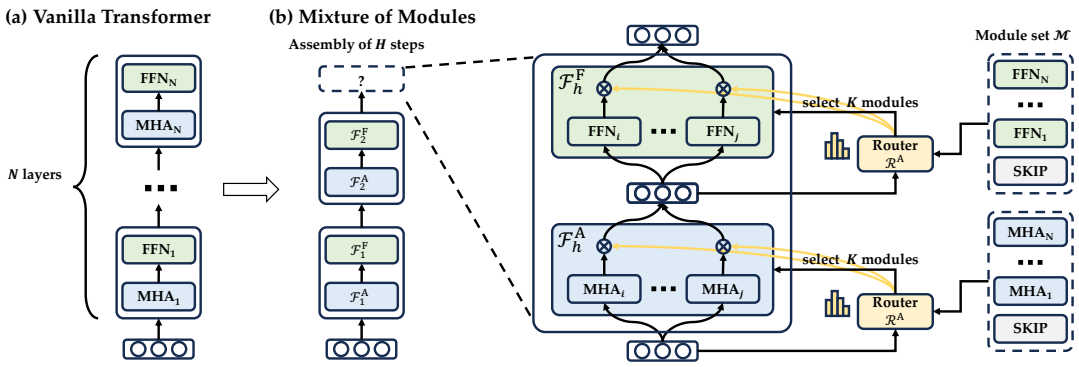

Figure 1: Mixture-of-Modules reinvents Transformers as dynamic assemblies of modules. In Figure (b), we illustrate the ongoing construction of an MoM model during the forward computation. The assembly lasts H rounds, with the current illustration showcasing progress in the third round. For each token, routers select the best K attention modules, denoted as m_k^{A}, and the best K feed-forward network modules, denoted as m_k^{F}, from a module set $\mathcal{M}$ (including "SKIP" modules). These selected modules collectively constitute assembled modules $\mathcal{F}^{\mathrm{A}}$ and $\mathcal{F}^{\mathrm{F}}$, which are then appended to the existing computation graph. Detailed notations are presented in §3.

As a typical approach in dynamic depth, *Early-exiting* (Graves, 2016; Figurnov et al., 2017; Schuster et al., 2022) accelerates model inference through terminating forward computation at intermediate layers. The decision to exit often relies on confidence-based metrics (Elbayad et al., 2020; Varshney et al., 2023; Xin et al., 2020) or pre-determined strategies (Liu et al., 2020; Corro et al., 2023). With some degree of generalization, *Layer-skip* (Srivastava et al., 2015; Wang et al., 2018b; Bapna et al., 2020) represents a more adaptive variant of early-exiting, enabling certain layers to be skipped without terminating the entire forward computation. Existing works mainly facilitate it by training a router (Zeng et al., 2023; Raposo et al., 2024) or layer pruning (Yang et al., 2024; Kim et al., 2024). Finally, if we view parameter copying as a particular way to increase network depth with controlled model size, then some parameter sharing methods (Dehghani et al., 2019; Lan et al., 2020), wherein certain modules or layers share parameters, also fall in the dynamic depth group.

In terms of dynamic width, *Mixture-of-Experts* (MoE, (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022)) is a representative method. An MoE model conceptualizes an FFN module as an "expert" for storing knowledge. Comprising multiple such experts, an MoE layer replaces the traditional FFN layer within Transformers, aiming for superior performance in handling knowledge-related tasks. During forward computation, a router network dynamically assigns each token to the top K experts out of a total of N experts, thereby increasing the maximum network width by K times. Other dynamic width methods, such as CODA (Lei et al., 2024) and CoLT5 (Ainslie et al., 2023), use similar routing mechanism to select whether a token passes through a heavy or light pathway for not only each FFN layer but also each attention layer.

MoM breaks the depth-ordered paradigm followed by existing approaches when performing forward computation. It not only unifies a number of approaches described above but also offers a more flexible and learnable way to achieve conditional computation.

3 Methodology

The idea of Mixture-of-Modules (MoM) is inspired by the "society of mind" theory by Marvin Minsky (Minsky, 1986), which explains the true intelligence as certain and very special ways of combinations of simple and modular units (in the book, they are termed "agents"). In §3.1, we first provide an overview of MoM. Then, we detail the assembly of modules and the routers in §3.2 and §3.3. After that, we present the training procedure of MoM in §3.4. Finally in §3.5, we show that MoM unifies various techniques of dynamic computation allocation within Transformers as special cases.

3.1 Mixture-of-Modules (MoM)

Before delving into the details, we first give a brief description of the workflow of MoM. MoM views the construction of an H-depth transformer as an H-step iterative assembly process. In each assembly step, router $\mathcal{R}$ dynamically selects K modules from a module set $\mathcal{M}$ for each token. Then these selected modules are assembled guided by the assembling function ϕ. Formally, MoM can be de-

fined by a 5-tuple $< \mathcal{M}, \mathcal{R}, \phi, K, H >$.

$\mathcal{M}$ is the set that contains all possible modules, where modules are defined as atomic units that could be assembled. There are two types of modules in a Transformer model, *i.e.*, the multi-head self-attention module (`MHA`) and the feed-forward network module (`FFN`), denoted as m^{A} and m^{F}, respectively. In addition, denote the input hidden state of a Transformer layer as $\mathbf{x} \in \mathbb{R}^d$, we include a special module $m^{\mathrm{S}} : \mathbf{x} \mapsto \mathbf{x}$ in $\mathcal{M}$, which means the absence of an operation applied to the token, allowing for skipping one round of computation. Therefore, in MoM, we have:

$$\mathcal{M} = \{m_i^{\mathrm{A}}\}_{i=1}^{N_{\mathrm{A}}} \cup \{m_i^{\mathrm{F}}\}_{i=1}^{N_{\mathrm{F}}} \cup \{m^{\mathrm{S}}\}, \quad (1)$$

where N_{A} are N_{F} refer to numbers of `MHA`s and `FFN`s, respectively.

$\mathcal{R}$ is a router responsible for dynamically selecting appropriate modules from $\mathcal{M}$ and assembling them into the computation graph. We use distinct routers for `MHA`s and `FFN`s, denoted as $\mathcal{R}^{\mathrm{A}}$ and $\mathcal{R}^{\mathrm{F}}$ respectively. The output of the router $\mathcal{R}^{\mathcal{X}}$ is an $(N_{\mathcal{X}} + 1)$-dimensional distribution wherein each item represents the weight assigned to each module $m_i^{\mathcal{X}}$ as well as m^{S}. Formally:

$$\begin{aligned} &\mathcal{R}^{\mathcal{X}} : \quad \mathbf{x} \mapsto \mathbf{r}^{\mathcal{X}}, \\ &\mathbf{x} \in \mathbb{R}^d, \quad \mathbf{r}^{\mathcal{X}} \in \mathbb{R}^{N_{\mathcal{X}}+1}, \quad \mathcal{X} \in \{\mathrm{A}, \mathrm{F}\}. \end{aligned} \quad (2)$$

An MoM model is dynamically assembled step by step. In each step, based on the output of $\mathcal{R}_{\mathcal{X}}$, $K_{\mathcal{X}}$ modules are selected from $\mathcal{M}$ and assembled together. The assembly process lasts H steps, as detailed in §3.2. Further elaboration on these routers, including their architecture and the working pipeline, is deferred to §3.3. Notably, in MoM, dynamic assembly occurs at the token level, wherein each token is independently and dynamically assigned by routers to appropriate modules for processing.

3.2 Dynamic assembly of modules

We delve into how an MoM model is dynamically assembled. The construction is an iterative process where in the h-th step (i.e., the h-th layer of the model being constructed), we have the input $\mathbf{x}_h$. The subscript h is omitted when there's no ambiguity. The router $\mathcal{R}$ selects K modules with the largest routing weight. We denote the indices of the selected modules as $\mathcal{K}_{\mathcal{X}} = \{i | r_i \in \mathrm{TopK}(\mathbf{r}^{\mathcal{X}})\}$. Then the selected modules are assembled together through the assembling function ϕ. Formally,

$$\phi : \quad < \mathcal{M}, \mathcal{R}^{\mathcal{X}}, \mathbf{x}_h > \mapsto \ \mathcal{F}_h^{\mathcal{X}}, \quad \mathcal{X} \in \{\mathrm{A}, \mathrm{F}\}, \quad (3)$$

where $\mathcal{F}_h^{\mathcal{X}}$ represents the assembled modules. These assembled modules transform the input $\mathbf{x}_h$ into the output $\mathbf{x}_{h+1}$. We hope the role of the h-th step of MoM assembly is somewhat akin to the h-th Transformer block in the conventional sense. Therefore, we establish two rounds of routing and assembling in each assembly step: one for `MHA` and the other for `FFN`. The forward computation of MoM models at the h-th step assembly can be represented as:

$$\begin{aligned} \mathbf{u}_h &= \mathcal{F}_h^{\mathrm{A}}(\mathbf{x}_h) + \mathbf{x}_h, \\ \mathbf{x}_{h+1} &= \mathcal{F}_h^{\mathrm{F}}(\mathbf{u}_h) + \mathbf{u}_h. \end{aligned} \quad (4)$$

We employ Pre-norm in MoM, which normalizes the input before feeding to assembled modules $\mathcal{F}^{\mathcal{X}}$. The dynamic assembly process is depicted in Figure 1(b). We now introduce the detailed formalization for $\mathcal{F}^{\mathrm{A}}$ and $\mathcal{F}^{\mathrm{F}}$, respectively.

Assembly of attention modules ($\mathcal{F}^{\mathrm{A}}$). We begin by considering the scenario where the m^{S} module is not selected by routers. Suppose that an `MHA` module contains Z individual heads, then the assembly of K_{A} `MHA` modules (i.e., the computation process of $\mathbf{o} = \mathcal{F}^{\mathrm{A}}(\mathbf{x})$) is defined as:

$$\begin{aligned} \mathbf{o} &= \mathbf{a} \sum_{k \in \mathcal{K}_{\mathrm{A}}} r_k^{\mathrm{A}} \cdot \mathbf{W}_k^O, \\ \mathbf{r}^{\mathrm{A}} &= \mathcal{R}^{\mathrm{A}}(\mathbf{x}) = \left(r_1^{\mathrm{A}}, \ldots, r_k^{\mathrm{A}}, \ldots, r_{K_A}^{\mathrm{A}}\right), \\ \mathbf{a} &= \left(\mathbf{x} \sum_{k \in \mathcal{K}_{\mathrm{A}}} \mathbf{W}_{k,z}^V\right) \cdot \\ &\quad \mathtt{softmax}\left(\frac{(\mathbf{X} \sum_{k \in \mathcal{K}_{\mathrm{A}}} \mathbf{W}_{k,z}^Q)(\mathbf{X} \sum_{k \in \mathcal{K}_{\mathrm{A}}} \mathbf{W}_{k,z}^K)^\top}{\sqrt{d_{\text{head}}}}\right), \end{aligned} \quad (5)$$

where $\mathbf{X} \in \mathbb{R}^{L \times d}$ is the input representation of the sequence, $\mathbf{W}_{k,z}^Q, \mathbf{W}_{k,z}^K, \mathbf{W}_{k,z}^V \in \mathbb{R}^{d \times d_{\text{head}}}$, and $\mathbf{W}_k^O \in \mathbb{R}^{d_{\text{head}} \times d}$ are weight matrices with $d_{\text{head}} = d/Z$. When the m^{S} module is selected, the operation of $\mathcal{F}^{\mathrm{A}}$ only involves the remaining $K_{\mathrm{A}} - 1$ attention modules. It is worth noting that our modification to the standard attention mechanism does not affect the KV-cache technique.

Assembly of feed-forward networks ($\mathcal{F}^{\mathrm{F}}$). The assembly of $\mathcal{F}^{\mathrm{F}}$ is more modular, where the outputs of K_{F} modules are simply weighted and aggregated. When the m^{S} module is not selected,

$\mathcal{F}^{\mathrm{F}}$ can be formalized as follows:

$$\begin{aligned}\mathcal{F}^{\mathrm{F}}(\mathbf{u}) &:= \sum_{k\in\mathcal{K}_{\mathrm{F}}} r_k^{\mathrm{F}} \cdot m_k^{\mathrm{F}}(\mathbf{u}),\\ \mathbf{r}^{\mathrm{F}} &= \mathcal{R}^{\mathrm{F}}(\mathbf{x}) = \left(r_1^{\mathrm{F}}, \ldots, r_k^{\mathrm{F}}, \ldots, r_{K_F}^{\mathrm{F}}\right).\end{aligned} \tag{6}$$

When m^{S} is chosen, likewise, only $K_{\mathrm{F}} - 1$ FFNs form the $\mathcal{F}^{\mathrm{F}}$.

3.3 MoM router ($\mathcal{R}$)

In prior approaches, routing occurs as a one-step decision-making process within a layer. However, in MoM which possesses a dynamically constructed computation graph, each decision is interdependent with the preceding ones, influencing the entire forward computation. Consequently, the router in MoM necessitates an awareness of past decisions. To model such dependency, we employ a gated recurrent unit (GRU, (Cho et al., 2014)) as the backbone of routers. Two routers in MoM are identical in structure. At each assembly step, the GRU in the $\mathcal{R}^{\mathcal{X}}$ maintains an $\mathbf{s}_h^{\mathcal{X}}$ as the hidden state of the GRU network. This state is recurrently updated as follows:

$$\begin{aligned}\mathbf{s}_h^{\mathrm{A}} &= \mathsf{GRU}^{\mathrm{A}}(\mathbf{x}_h, \mathbf{s}_{h-1}^{\mathrm{A}}),\\ \mathbf{s}_h^{\mathrm{F}} &= \mathsf{GRU}^{\mathrm{F}}(\mathbf{u}_h, \mathbf{s}_{h-1}^{\mathrm{F}}).\end{aligned} \tag{7}$$

The weights assigned to each module by $\mathcal{R}^{\mathcal{X}}$ are computed as:

$$\begin{aligned}&\mathbf{r}^{\mathcal{X}} = \mathbf{W}_{\mathcal{X}}\mathbf{s}_h^{\mathcal{X}},\\ &\mathbf{W}_{\mathcal{X}} \in \mathbb{R}^{(N_{\mathcal{X}}+1)\times d}, \quad \mathcal{X} \in \{\mathrm{A}, \mathrm{F}\}.\end{aligned} \tag{8}$$

3.4 Training approach

A straightforward approach is to pre-train an MoM model initialized from scratch. This approach, however, suffers from a degeneration issue, as the learned functions of modules become homogeneous, making router training challenging. To address the issue, we propose a two-phase training approach. In the first phase, we pre-train a vanilla Transformer where modules acquire distinct functionalities. Then, in the second phase, we initialize the module set $\mathcal{M}$ with the pre-trained modules and initialize the routers from scratch. Subsequently, we continue training both modules and routers using the same data and objective as in the first phase.

3.5 MoM as a unified framework

A compelling property of MoM is that it unifies a wide range of Transformer-based dynamic computation allocation architectures. With specific configurations, layer-skip (e.g., early-exiting, mixture-of-depths, etc.), parameter sharing, and mixture-of-experts can be viewed as special cases.

Layer-skip. The key idea is to skip layers according to certain criteria which can either be defined heuristically (Liu et al., 2024) or learned from data (Zeng et al., 2023; Raposo et al., 2024). Within the MoM framework, layer-skip can be formulated as a special cluster of assembly functions ϕ, namely:

$$\phi_{\text{layer-skip}}(\mathcal{M}, \mathcal{R}^{\mathcal{X}}, \mathbf{x}_h) = \begin{cases} m_h^{\mathcal{X}} & \text{if } c_{\text{skip}}(h) = 1\\ m^{S} & \text{if } c_{\text{skip}}(h) = 0\end{cases}, \tag{9}$$

where $c_{\text{skip}}(\cdot)$ is the criterion that decides whether to skip the h-th layer or not. Note that the technique of early-exiting (Graves, 2016; Figurnov et al., 2017) can be viewed as a special case of layer-skip, where once a layer is skipped, all subsequent layers will be skipped too.

Parameter sharing. We consider parameter sharing that shares weights across modules and does not involve reparameterization techniques. Under this restriction, the sharing paradigm can be defined as a criterion function $c_{\text{share}} : i \mapsto j\ (j \le i)$, representing using the same weights for module $m_i^{\mathcal{X}}$ and module $m_j^{\mathcal{X}}$. Within MoM, parameter sharing can be formulated as:

$$\phi_{\text{parameter-sharing}}(\mathcal{M}, \mathcal{R}^{\mathcal{X}}, \mathbf{x}_h) = m^{\mathcal{X}}_{c_{\text{share}}(h)}. \tag{10}$$

Mixture-of-Experts. MoE splits FFN into experts, and the experts are not shared across different layers. In MoE, routing is only performed on FFN modules, thus the computation of MHA is the same as that of a vanilla Transformer. The assembly function for MoE can be written as:

$$\phi_{\text{MoE}}(\mathcal{M}, \mathcal{R}^{\mathcal{X}}, \mathbf{x}_h) = \begin{cases} m_h^{\mathrm{A}} & \text{if } \mathcal{X} = \mathrm{A}\\ \phi_{\text{MoM}}(\mathcal{M}_h, \mathcal{R}^{\mathrm{F}}, \mathbf{x}_h) & \text{if } \mathcal{X} = \mathrm{F}\end{cases}, \tag{11}$$

where $\mathcal{M}_h = \{m_{h,i}^{\mathrm{F}}\}_{i=1}^{N_{\mathrm{F}}}$ is the collection of experts for layer h.

Figure 2 illustrates the forward computation process across different methods, offering an intuitive presentation of the versatility and universality in MoM.

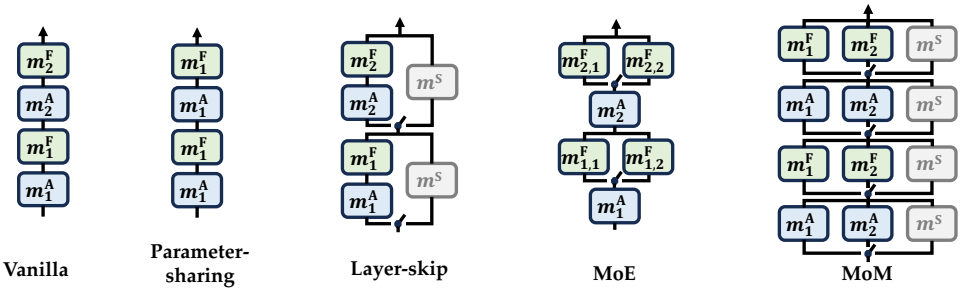

Figure 2: Visualization of forward computation of five models, where each consists of only two layers just for demonstration purposes. The switch icon symbolizes the selective execution of one (in Layer-skip) or more (in MoE and MoM) subsequent computation pathways.

4 Experiments

4.1 Experimental setup

We implement language models with MoM since language modeling is a challenging task requiring both language understanding and generation ability, thereby effectively evaluating MoM and baselines. Below, we elaborate on the implementation details of MoM, baselines, and evaluation setups.

Implementation details. We conduct experiments on three model scales, which we denote as MoM-small, MoM-medium, and MoM-large. These models contain 112M, 346M, and 774M parameters respectively. Detailed configurations can be found in Appendix A. The vanilla transformer used for initializing MoM are official GPT2 checkpoints downloaded from HuggingFace[2]. K and H represent two hyper-parameters of MoM. We denote a configuration where $K = a, H = b$ as KaHb. If the skip module is included in $\mathcal{M}$, we append the suffix S to KaHb.

In the two-phase training, we exploit OpenWebText (Gokaslan and Cohen, 2019) as the pre-training dataset, and pre-process the data with the same pipeline as nanoGPT (karpathy, 2023). OpenWebText contains 9 billion tokens after tokenization, from which 4 million tokens are randomly sampled as the validation set. The training sequence length for every input is $1,024$. We set the learning rate to 1e-3 with a warm-up ratio of 0.1 throughout the two phases, and do not use dropout. All models are trained on $8 \times$ A100 GPUs with a total batch size of 8×64. Two training phases require 20k and 10k optimization steps, respectively.

In practice, considering the large search space characterized by H and K, we confine the architecture search space to a practical scale with a "chunking" strategy. An MoM is divided into several chunks. Each chunk is independent and parameterized with identical H and K. We present a detailed description and specific configuration in Appendix A, and an empirical analysis of the efficacy of chunking in Appendix B.

Baselines. In addition to the vanilla Transformer model (Radford et al., 2019), the following models (or methods) are also employed as baselines: (1) **MoD** (Raposo et al., 2024): a layer-skip method proposed recently that dynamically routes around Transformer blocks.[3] (2) **MoE**: the mixture-of-experts architecture utilized by Mixtral. We implement the model with the open-sourced code.[4] (3) **MoE (share)**: a variant of MoE in which all layers share the same set of experts. We involve this model as a baseline because unlike the standard MoE that has more parameters, MoE-share has the same number of parameters with the vanilla Transformer model, making the comparison more fair. Moreover, it also sheds light on how well the MoE architecture can utilize a fixed budget of modules.

Note that all the above-mentioned methods are special cases within the MoM framework with various configurations. Furthermore, we explore a wide range of MoM instances defined by KaHbS, where $a \leq 4$ and $b \leq 6$. We examine all MoM instances within the search space (as detailed in §4.3), and spotlight three distinct models for comparison against other baselines:

- **MoM$_{\mathbf{P}}$** (K2H6S) represents a **P**erformant MoM model after tuning K and H.

[2] https://huggingface.co/openai-community/gpt2

[3] As official code is unavailable until the submission, we follow the paper to implement MoD ourselves.

[4] https://github.com/huggingface/transformers/blob/v4.36.1/src/transformers/models/mixtral/modeling_mixtral.py

Methods ϕ	MoM Config (KaHb)	Parameter Count	Computation Cost (TFLOPs)	Memory Cost (Gb)	Validation Loss	Validation Perplexity	GLUE (Average)	XSUM (Average)
			small					
GPT2	K1H4	122M	2.92	2.98	3.10	22.22	75.32	14.26
MoD	K1H4S	122M	-	-	3.22	25.11	72.24	9.71
MoE	K2H4	283M	3.81 (+30.5%)	2.98 (+0.0%)	3.07	21.18	77.25	14.18
MoE (share)	K2H4	122M	3.81 (+30.5%)	2.98 (+0.0%)	3.14	23.41	75.82	14.15
MoM_E	K3H1S	122M	2.45 (-16.1%)	2.45 (-17.8%)	3.16	23.59	75.92	14.17
MoM_I	K3H2S	122M	3.49 (+19.5%)	2.63 (-11.7%)	3.03	20.79	77.81	14.24
MoM_P	K2H6S	122M	5.04 (+72.6%)	3.34 (+12.1%)	**2.98**	**19.59**	**78.22**	**15.19**
			medium					
GPT2	K1H4	346M	8.28	4.74	2.81	16.69	80.49	18.14
MoD	K1H4S	346M	-	-	2.99	19.82	76.17	14.81
MoE	K2H4	921M	11.37 (+37.3%)	4.74 (+0.0%)	2.80	16.53	80.47	17.75
MoE (share)	K2H4	346M	11.37 (+37.3%)	4.74 (+0.0%)	2.82	16.81	80.35	17.59
MoM_E	K3H1S	346M	6.80 (-17.9%)	3.33 (-29.7%)	2.83	16.91	80.41	17.11
MoM_I	K3H2S	346M	10.20 (+23.2%)	3.80 (-19.8%)	2.77	15.89	81.03	18.66
MoM_P	K2H6S	346M	16.23 (+96.0%)	5.69 (+20.0%)	**2.72**	**15.18**	**81.93**	**19.30**
			large					
GPT2	K1H4	774M	17.76	7.20	2.66	14.33	84.47	20.35
MoD	K1H4S	774M	-	-	2.81	16.62	81.49	18.62
MoE	K2H4	2100M	25.43 (+43.2%)	7.20 (+0.0%)	2.64	14.17	84.43	20.63
MoE (share)	K2H4	774M	25.43 (+43.2%)	7.20 (+0.0%)	2.65	14.22	83.83	20.39
MoM_E	K3H1S	774M	14.84 (-16.4%)	4.13 (-42.6%)	2.66	14.50	83.39	20.46
MoM_I	K3H2S	774M	20.31 (+14.5%)	5.15 (-28.5%)	2.64	13.92	84.49	21.73
MoM_P	K2H6S	774M	36.07 (+103.1%)	9.24 (+28.3%)	**2.60**	**13.21**	**85.90**	**22.36**

Table 1: Comprehensive comparison between MoMs and baselines. We highlight the best results in bold and underline the second-best results. Appendix C includes the detailed performance on GLUE and XSUM.

- $\textbf{MoM}_\textbf{E}$ (K3H1S) significantly enhances **E**fficiency compared to vanilla Transformers, while maintaining acceptable performance.
- $\textbf{MoM}_\textbf{I}$ (K3H2S) serves as a midpoint in configurations between the two preceding models. This model is positioned between the efficiency and the performance. We aim to highlight MoM's feature of performance and efficiency **I**nterpolation through configuration interpolation.

Evaluation settings. We employ GLUE benchmark (Wang et al., 2018a) to evaluate the language understanding ability of MoMand XSUM (Narayan et al., 2018a) to evaluate the text generation ability. All models are fine-tuned with a learning rate of 2e-5. The sequence is 128 for GLUE and 1024 for XSUM. For smaller GLUE sub-datasets (CoLA, STS-B, MRPC, and RTE), we set the batch size to 32 and train for 3 epochs. For larger datasets (MNLI, QNLI, QQP, and SST-2), we utilize a batch size of 64 and perform training for a total of $8,000$ gradient steps. For XSUM, we set the batch size to 64 and train for 3 epochs. For efficiency evaluation, we report inference TFLOPs and memory usage. TFLOPs are calculated using DeepSpeed FLOPs profiler (DeepSpeed, 2023) and memory consumption is calculated with PyTorch toolkits (pytorch, 2023).

4.2 Main results

Table 1 reports the evaluation results. Our analysis yields the following conclusions:

MoM unleashes the potential of Transformers and our initial motivation is confirmed. When maintaining the number of parameters, MoM_P is characterized by the deepest computation graph (H). Across all model scales, MoM_P consistently outperforms all baselines on both GLUE and XSUM by significant margins. The enhanced performance of MoM_P validates our initial motivations: (1) the traditional depth-ordered layer organization is sub-optimal; (2) improvements can be realized through two key modifications to the computation graph, including *dynamic module organization* and *improved parameter utilization*.

MoM_E is characterized by its minimum depth (H). By strategically selecting appropriate modules at each assembly step, MoM_E strives to reduce memory and computation costs while main-

taining performance. Although on medium and large scales, MoM$_{\text{E}}$ is slightly surpassed by a vanilla Transformer, it outperforms MoD, another efficiency-driven method by a large margin.

Besides, we observe that MoM$_{\text{I}}$ archives a decent performance by slightly outperforming vanilla GPT2. Comparing to vanilla GPT2, MoM$_{\text{I}}$ consumes no more than 25% extra computation but save at most 28.5% memory across all scales, indicating that MoM$_{\text{I}}$ achieves a good balance between performance and efficiency.

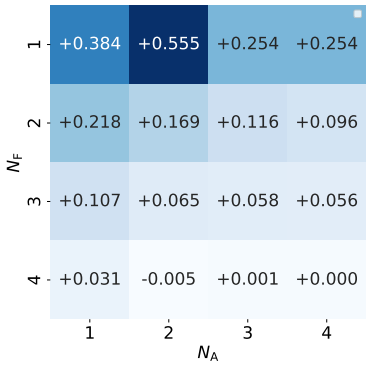

Figure 3: How validation loss varies with respect to N_{A} and N_{F}, comparing to MoM (medium) with $N_{\text{A}} = N_{\text{F}} = 4$.

MoM models provide insights into the over-parameterization issue quantitatively. We develop a series of MoM models, each defined by different pairs of $(N_{\text{A}}, N_{\text{F}})$, with both values not exceeding 4. We assess the validation loss increase for each model relative to the benchmark model where $N_{\text{A}} = 4$ and $N_{\text{F}} = 4$, as illustrated in Figure 3. In this experiment, we set K equal to the number of modules to make full use of the module parameters. Interestingly, the FFN and MHA modules exhibit different degrees of redundancy. Specifically, when N_{F} is fixed and the number of MHAs is gradually reduced, a significant increase in loss is not observed until N_{A} is reduced from 2 to 1, suggesting considerable redundancy in the MHAs of Transformers. In contrast, when fixing N_{A} and reducing the number of FFNs gradually, each time of removing an FFN leads to evident loss increase, indicating FFNs are less over-parameterized. These quantitative findings align with previous research suggesting that the parameterization of attention can be simplified to enhance efficiency while maintaining performance (DeepSeek-AI, 2024; Shazeer, 2019).

As the parameter size scales up, MoM models enjoy consistent gains in both performance and efficiency. When we look into the difference across different scales, we observe that (1) the performance gain of MoM is stable; (2) MoM$_{\text{E}}$-medium and MoM$_{\text{E}}$-large exhibit more significant reductions in resource costs comparing to MoM$_{\text{E}}$-small. These observations across different scales reinforce our previous motivation: Transformers are over-parameterized, which becomes more evident as the model size increases.

4.3 Insights from hyper-parameter search

Figure 4 shows how the validation loss for MoM-small and MoM-medium varies with respect to different settings of K and H ($K \in \{1, 2, 3, 4\}$, $H \in \{1, 2, 3, 4, 5, 6\}$). From this experiment, we have the following observations and insights: (1) allowing more modules to be assembled at each step (i.e., larger K) and more rounds of assembling actions (i.e., larger H) generally leads to better performance, indicating that *Transformer-based models benefit from a larger computation graph even if the parameter size remains the same*. However, (2) the benefits of increasing K and H become marginal when $K > 2$ and $H > 1$. Comparing K3H1 to K2H6, we can see that the validation loss is comparable, while K3H1 performs slightly worse on downstream tasks as discussed in §4.2. However, K3H2 improves efficiency by flattening the depth, making it a good choice that balances performance and efficiency. Flattening modules from different depths to the same depth cancels computation dependencies of each other. This characteristic brings an extra benefit because the computation of modules from the same depth can be parallelized. This technique has been validated and adopted in MoE applications (Fedus et al., 2022; Lepikhin et al., 2021) (called expert parallelism) and can be easily extended to further accelerate MoM (K3H2).

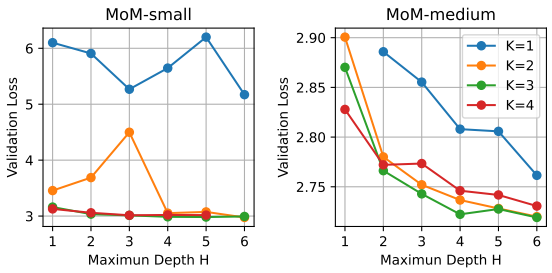

Figure 4: Validation loss for MoM-small and MoM-medium under different settings of K and H.

5 Conclusions

In this work, we propose Mixture-of-Modules (MoM), a novel architecture that reinvents transformers as a collection of individual modules and the dynamic assembly process conducted with these modules. This novel view offers us an opportunity to explore a wide range of different configurations of model architecture and unify a series of transformer variants. With exhaustive experiments, we not only validate the effectiveness of MoM by both significant efficiency and performance gains but also reach new insights about Transformers.

Limitations

Our current design of the router still has room for improvement. Unlike MoE wherein the router makes one-time decisions about which experts to select, the router of MoM is responsible for conducting multi-step decision-making. In this scenario, instructing the router to make correct decisions continuously is a hard question since the decision space grows exponentially with the increase of assembly steps. The current implementation has not considered this question and has not explicitly encouraged or discouraged the router to make some choices, thus, we are not sure whether the learned routing decisions are optimal or not. In the future, we will explore using techniques like reinforcement learning or neural architecture search to design more sophisticated routers.

References

Andrea Agostinelli, Timo I. Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, Matt Sharifi, Neil Zeghidour, and Christian Frank. 2023. Musiclm: Generating music from text. *Preprint*, arXiv:2301.11325.

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, et al. 2023. Colt5: Faster long-range transformers with conditional computation. *arXiv preprint arXiv:2303.09752*.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2020. Controlling computation versus quality for neural sequence models. *Preprint*, arXiv:2002.07106.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *Preprint*, arXiv:1308.3432.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *Preprint*, arXiv:2303.12712.

Kyunghyun Cho and Yoshua Bengio. 2014. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *Preprint*, arXiv:1406.7362.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Luciano Del Corro, Allie Del Giorno, Sahaj Agarwal, Bin Yu, Ahmed Awadallah, and Subhabrata Mukherjee. 2023. Skipdecode: Autoregressive skip decoding with batching and caching for efficient llm inference. *Preprint*, arXiv:2307.02628.

Andrew Davis and Itamar Arel. 2014. Low-rank approximations for conditional feedforward computation in deep neural networks. *Preprint*, arXiv:1312.4461.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

DeepSpeed. 2023. Deepspeed flops profiler.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive transformer. *Preprint*, arXiv:1910.10073.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1).

Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. 2017. Spatially adaptive computation time for residual networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1790–1799.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf,

Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *Preprint*, arXiv:1603.08983.

karpathy. 2023. nanogpt, the simplest, fastest repository for training/finetuning medium-sized gpts.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. 2024. Shortened llama: A simple depth pruning for large language models. *Preprint*, arXiv:2402.02834.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Zhao, Yuexin Wu, Bo Li, et al. 2024. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*.

Yijin Liu, Fandong Meng, and Jie Zhou. 2024. Accelerating inference in large language models with a unified layer skipping strategy. *Preprint*, arXiv:2404.06954.

Yijin Liu, Fandong Meng, Jie Zhou, Yufeng Chen, and Jinan Xu. 2020. Faster depth-adaptive transformers. In *AAAI Conference on Artificial Intelligence*.

Ang Lv, Yuhan Chen, Kaiyi Zhang, Yulong Wang, Lifeng Liu, Ji-Rong Wen, Jian Xie, and Rui Yan. 2024. Interpreting key mechanisms of factual recall in transformer-based language models. *Preprint*, arXiv:2403.19521.

Ali Madani, Ben Krause, Eric R. Greene, Subu Subramanian, Benjamin P. Mohr, James M. Holton, Jose Luis Olmos, Caiming Xiong, Zachary Z. Sun, Richard Socher, James S. Fraser, and Nikhil Naik. 2023. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, 41(8):1099–1106.

Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. 2023. The hydra effect: Emergent self-repair in language model computations. *Preprint*, arXiv:2307.15771.

M. Minsky. 1986. *The Society of Mind*. Touchstone book. Simon and Schuster.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018a. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018b. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

William Peebles and Saining Xie. 2023. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4195–4205.

pytorch. 2023. Pytorch profiler.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *Preprint*, arXiv:2404.02258.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. In

Advances in Neural Information Processing Systems, volume 35, pages 17456–17472. Curran Associates, Inc.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *Preprint*, arXiv:1911.02150.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Abubakr Babiker, Nathanael Schärli, Aakanksha Chowdhery, Philip Mansfield, Dina Demner-Fushman, Blaise Agüera y Arcas, Dale Webster, Greg S. Corrado, Yossi Matias, Katherine Chou, Juraj Gottweis, Nenad Tomasev, Yun Liu, Alvin Rajkomar, Joelle Barral, Christopher Semturs, Alan Karthikesalingam, and Vivek Natarajan. 2023. Large language models encode clinical knowledge. *Nature*, 620(7972):172–180.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *Preprint*, arXiv:1505.00387.

Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. 2023. Accelerating llama inference by enabling intermediate layer decoding via instruction tuning with lite. *Preprint*, arXiv:2310.18581.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018a. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018b. Skipnet: Learning dynamic routing in convolutional networks. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, page 420–436, Berlin, Heidelberg. Springer-Verlag.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024. Laco: Large language model pruning via layer collapse. *ArXiv*, abs/2402.11187.

Dewen Zeng, Nan Du, Tao Wang, Yuanzhong Xu, Tao Lei, Zhifeng Chen, and Claire Cui. 2023. Learning to skip for language modeling. *Preprint*, arXiv:2311.15436.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc.

A More implementation details

Table 2 lists the configuration of MoM-small/medium/large.

In practice, we segment MoM into equally-sized chunks, each containing 4 `MHA` modules and 4 `FFN` modules, namely $N = 4$. Within each chunk, we execute the MoM assembly process as presented in §3. We restrict the search space of each chunk by setting $K \leq 4$ and $H \leq 6$, which results in $4 \times 6 = 24$ combinations in total.

Then we elaborate on the initialization of chunked MoM. We take the 8-layer vanilla transformer as an example. We slice it this way: the bottom/top 2 layers remain the same, and modules in the middle 4 layers form a MoM block, wherein we conduct the iterative assembly process. We denote this chunking strategy as `[1-1-4-1-1]` where "1" represents the standard Transformer block and "4" represents a chunk whose N equals to 4. Empirically, we find this setting to be stable across various choices of KaHb. A detailed experimental analysis of different chunking strategies can be found in Appendix B. Similarly, for MoM-small, MoM-medium and MoM-large, we use the chunking strategies of `[1-1-4-1-4-1]`, `[1-1-1-4-1-4-1-4-1-4-1-1]`, and `[1-4-1-4-1-4-1-4-1-4-1-4-1]`, respectively.

B Chunks

In this section, we study the impact of different chunking strategies on MoM performance. This experiment is conducted on an 8-layer MoM. Except for `[1-1-4-1-1]`, we include several alternatives: `[4-4]` (two successive MoM block with $N = 4$), `[1-6-1]` (the top and bottom one layer unchanged, modules in the middle 6 layers form a MoM with $N = 6$), and `[8]` (all modules forms a big MoM with $N = 8$). Table 3 shows the results of different chunking strategies. When $K = 1$, strategies other than `[1-1-4-1-1]` exhibit unstable training curves and bad performance. This is because the routers need to make multi-step decisions in the search space. A larger search space (the increase of N) and more assembly steps (the increase of H) all lead to a harder task for the routers to find the correct path in the search space. Things are much better when $K = 2$, where all strategies converge quite well. This experiment demonstrates the necessity of manually restricting the search space of MoM so that the decision-making burden for the routers would be relieved.

C More downstream evaluation results

Table 4 presents the evaluation results for each subtask of GLUE across different models and Table 5 presents the detailed results on XSUM dataset.

Table 6 presents the zero-shot evaluation results for each subtask of SuperGLUE (Wang et al., 2019).

D Router analysis

D.1 Architecture design

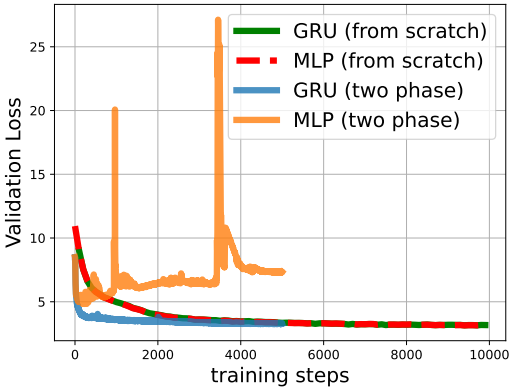

Figure 5: Training curves of MoM-small (K1H4) with {GRU, MLP} routers.

We study the implementation of a key component in MoM: the routers. We substitute the GRU within the router with a simple two-layer MLP, eliminating the interaction among router decision states. Our exploration of the router's impact involves two setups: (a) initializing MoM from scratch, and (b) employing the two-phase training approach. Here are some intriguing results. As depicted in Figure 5, when initializing from scratch, both router structures exhibit nearly identical loss curves. However, under setting (b), training MoM with the MLP router becomes unstable marked by spikes in gradient magnitude throughout the training. This instability suggests the router's inability to establish a consistent assembly plan for tokens. When initializing from scratch, the required capability may not be learned efficiently by these modules, as they often develop homogeneous functionalities that waste the parameters. Conversely, in setting (b), where modules are initialized with

	MoM-small	MoM-medium	MoM-large
Initialization model	GPT2-small	GPT2-medium	GPT2-large
Hidden size	768	1024	1280
Total number of FFN/MHA modules	12	24	36
Number of attention heads	12	16	20
Max sequence length	1204	1024	1024
Vocabulary size	50257	50257	50257

Table 2: Model configurations for MoM-small/medium/large.

Chunking Strategies	MoM Config	Val. Loss
[1-1-4-1-1]	K1H4S	3.27
[1-1-4-1-1]	K2H4S	3.22
[1-6-1]	K1H6S	3.45
[1-6-1]	K2H6S	3.21
[8]	K1H8S	4.63
[8]	K2H8S	3.23
[4-4]	K1H4S	5.59
[4-4]	K2H4S	3.22

Table 3: Applying different chunking strategies on an 8-layer MoM. These models follow the same two-phase training procedure and the total training steps of the second phase is 5k.

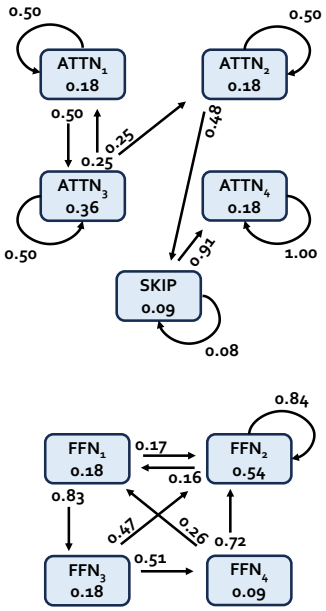

Figure 6: Routing patterns of MoM-medium.

specialized functions, the optimization progresses smoothly and converges quickly.

D.2 Learned router patterns

We are curious whether the router follows specific patterns when choosing and assembling modules. We visualize the transition probabilities between modules (Figure 6) to answer this. The first observation is that the router does not degrade into simply memorizing the original shallow-to-deep order but jumps across modules as expected. For example, a common routing path in Figure 6 is $(2, 2, 1, 3, 4, 2)$ for `FFN` modules and $(3, 1, 3, 2, S, 4)$ for `MHA` modules (number represents the module index). Another observation is that the loads of different modules are imbalanced. In some cases, specific modules are hardly used. Unlike MoE, which uses an auxiliary loss to balance the loads across different experts (Fedus et al., 2022), we do not see a positive effect by adding a balance loss to MoM. Adding regularization alleviates the imbalance issue at the cost of performance degradation (by increasing validation perplexity by 1.8 points). We posit an intuitive explanation: within the language model framework, tasks that can be decomposed into numerous subtasks may exhibit various levels of difficulty. Consequently, some subtasks necessitate more engagement of modules with specific processing capabilities, thus contributing to the observed imbalance.

Method	SST-2	COLA	MRPC	QQP	QNLI	RTE	MNLI-(m/mm)	STS-B	average
					small				
GPT2	92.09	26.27	85.11	83.09	87.55	62.45	79.90/78.74	82.67	75.32
MoD	87.73	21.66	81.43	82.01	81.79	64.62	75.96/75.22	79.76	72.24
MoE	89.68	45.87	82.05	84.09	85.94	65.52	79.46/78.74	83.87	77.25
MoE (share)	89.53	37.29	82.01	83.49	85.21	64.94	78.49/77.99	83.43	75.82
MoM_{E}	90.83	31.83	82.92	83.49	84.79	69.68	78.52/77.40	83.84	75.92
MoM_{I}	90.02	47.21	82.90	84.07	85.34	66.43	79.82/79.72	84.79	77.81
MoM_{P}	91.40	46.16	82.68	84.62	86.49	67.51	80.85/80.18	84.06	78.22
					medium				
GPT2	94.15	48.18	86.00	85.94	90.41	64.98	84.02/83.92	86.77	80.49
MoD	89.45	37.43	84.97	84.21	84.88	64.26	78.94/77.94	83.47	76.17
MoE	91.86	49.20	85.51	86.39	89.09	68.12	84.44/83.49	86.17	80.47
MoE (share)	92.78	49.44	84.56	86.40	89.35	66.79	84.00/83.39	86.42	80.35
MoM_{E}	91.40	48.48	87.48	86.92	89.11	67.51	83.73/83.01	86.06	80.41
MoM_{I}	92.46	50.19	87.31	86.43	89.35	68.94	84.44/83.46	86.66	81.03
MoM_{P}	92.88	53.61	87.64	86.64	89.75	71.06	84.69/83.98	87.14	81.93
					large				
GPT2	94.15	60.04	88.74	87.88	91.89	75.45	86.80/85.98	89.30	84.47
MoD	91.51	52.86	87.29	87.20	89.57	67.87	85.07/84.38	87.66	81.49
MoE	94.32	61.19	88.54	88.36	92.04	71.68	87.38/86.94	89.45	84.43
MoE (share)	93.88	61.43	88.12	88.17	91.56	68.31	87.01/86.73	89.28	83.83
MoM_{E}	93.64	59.26	88.25	87.59	91.29	72.23	85.20/84.78	88.26	83.39
MoM_{I}	93.69	62.25	89.19	88.12	92.36	74.98	87.22/86.78	89.90	84.94
MoM_{P}	94.42	64.49	89.56	88.68	92.94	77.69	87.68/86.98	90.70	85.90

Table 4: Detailed evaluation on the GLUE benchmark. We follow the previous evaluation setting (Radford et al., 2018), for SST-2, QNLI, RTE, and MNLI, we report accuracy as the metric. For MRPC and QQP, we report the F1 score. For STS-b, we report the combined score of Pearson correlation and Spearman correlation.

Method	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-AVG
		small		
GPT2	20.8	5.05	16.92	14.26
MoD	14.45	2.89	11.80	9.71
MoE	20.56	5.26	16.71	14.18
MoE (share)	20.64	5.17	16.64	14.15
MoM_{E}	20.62	4.98	16.91	14.17
MoM_{I}	20.50	5.31	16.90	14.24
MoM_{P}	21.73	6.14	17.72	15.19
		medium		
GPT2	25.07	8.35	21.00	18.14
MoD	20.89	6.04	17.52	14.81
MoE	24.58	8.20	20.49	17.75
MoE (share)	24.36	8.19	20.22	17.59
MoM_{E}	24.56	7.04	19.72	17.11
MoM_{I}	26.29	8.21	21.47	18.66
MoM_{P}	26.61	9.05	22.24	19.30
		large		
GPT2	28.16	9.92	22.98	20.35
MoD	26.08	8.38	21.41	18.62
MoE	28.54	9.89	23.47	20.63
MoE (share)	28.47	9.47	23.23	20.39
MoM_{E}	28.48	9.65	23.24	20.46
MoM_{I}	30.91	10.21	24.08	21.73
MoM_{P}	31.38	10.77	24.93	22.36

Table 5: Detailed evaluation on the XSUM dataset. ROUGE is employed as the evaluation metrics.

Method	BoolQ	CB	COPA	MultiRC	ReCoRD	WiC	WSC	average
				small				
GPT2	48.72	41.07	62.00	53.01	70.93	49.22	43.27	52.60
MoM_{E}	59.91	41.07	62.00	57.30	59.46	50.00	36.54	52.32
MoM_{P}	57.80	42.86	66.00	55.88	61.61	51.41	36.54	53.16
				medium				
GPT2	58.59	42.86	69.00	52.29	79.38	50.00	41.35	56.21
MoM_{E}	57.74	39.29	68.00	52.56	73.86	50.00	36.54	54.00
MoM_{P}	59.36	44.64	70.00	55.32	77.36	50.00	36.54	56.17

Table 6: Detailed evaluation on the SuperGLUE benchmark. For this evaluation, we use the Language Model Evaluation Harness library (Gao et al., 2024) with default hyperparameters and evaluation settings.